LSM树中基于热度预测的异构布隆过滤器方案

俞加平,陈华辉,钱江波,董一鸿
(宁波大学信息科学与工程学院,浙江宁波 315211)

摘 要: 日志结构合并(Log-Structured-Merge,LSM)树中常使用布隆过滤器减少无效磁盘I/O.但是用户无法无限制地细化布隆过滤器的粒度,原因是在一些数据量庞大而数据项较小的工作流中,这些元数据需要占用大量存储空间.其次在一些内存受限的环境下,内存缓冲区无法容纳更多的过滤器数据,造成缓冲区与磁盘的频繁数据交换.针对上述问题本文提出LSM树中的异构布隆过滤器方案,在LSM树的每一层维护热度预测模型,新生成的SSTable通过预测的热度来分配不同粒度的布隆过滤器,然后使用特定缓存管理方案来维护缓存中的过滤器数据并处理工作流热度发生改变的情况.实验证明,本文的方案在保持相同外存占用与内存消耗的情况下,读取吞吐量比采用原始LSM树结构的LevelDB提升22%~53%.

关键词: 日志结构合并树;键值存储;读取性能;布隆过滤器;存储管理;热度预测
中图分类号: TP392 **文献标识码:** A **文章编号:** 0372-2112(2021)11-2090-05
电子学报URL: http://www.ejournal.org.cn **DOI:** 10.12263/DZXB.20200945

A Heterogeneous Bloom Filter Scheme in LSM Tree Based on Hotness Prediction

YU Jia-ping, CHEN Hua-hui, QIAN Jiang-bo, DONG Yi-hong
(*College of Information Science and Engineering, Ningbo University, Ningbo, Zhejiang* 315211, *China*)

Abstract: Bloom filters are often used in the LSM(Log-Structured-Merge) tree to reduce unnecessary disk I/O. However, users cannot refine the granularity of Bloom filters infinitely, because in some workflows with huge data volume and small data items, these metadata require a lot of storage space. Secondly, in some memory-constrained environments, the cache cannot accommodate more filter data, resulting in frequent data exchange between memory and disk. In view of the above problems, we propose a heterogeneous Bloom filter scheme in LSM tree. The hotness prediction model is maintained at each layer of the LSM tree, and the newly generated disk components are distributed with different granularity according to the predicted hotness. And a specific cache management scheme is used to maintain the filter data in memory cache and deal with changes in workflow hotness. It is experimentally proven that the proposed scheme improves read throughput by 22%~53% over LevelDB with the original LSM tree structure while maintaining the same storage occupation and memory consumption.

Key words: log-structured-merge-tree; key-value store; read performance; Bloom filter; storage management; hotness prediction

1 引言

日志结构合并(Log-Structured-Merge,LSM)树[1]在现代NoSQL数据库存储层中应用广泛,它将更新缓存在主存中,然后使用顺序I/O持久化到磁盘并基于一定的策略进行合并.这种存储方式具有更优越的写性能,更高的空间利用率,更简单的并发控制和数据恢复等优点[2,3],契合工作流多变而繁重的大数据时代.

布隆过滤器[4]常用于构成LSM树中SSTable的辅助数据结构,它伴随SSTable的形成而产生.当查询一条记录时,LSM树会对一部分SSTable进行检查,在进行I/O之前会利用布隆过滤器判断某个键是否存在.布隆过滤器可能存在假阳性的情况,这将导致没有必要的I/O.通过概率分析可知布隆过滤器的假阳性率为$(2^{-\ln 2})^b$,b表示分配给每个键的位数(本文称b为布隆过

收稿日期:2020-08-27;修回日期:2021-04-14;责任编辑:覃怀银
基金项目:国家自然科学基金(No. 61572266, No. 61472194);浙江省自然科学基金(No. LZ20F020001,No. LY20F020009)

滤器的粒度,b越大,粒度越细). 显然可以通过增加b来降低假阳性率,但是会增加内外存负担.

实际应用中的数据的访问热度存在着很大的差异,因而若过滤器能对热度高的数据有更低的假阳性率,减少这部分数据的I/O,则能使系统在过滤器存储总开销不增加的前提下有更好的I/O性能. 基于此考虑,本文提出一种LSM树中数据的热度预测框架并设计异构布隆过滤器方案(Heterogeneous Bloom Filter Scheme in LSM-tree, HBF-LSM). HBF-LSM充分利用读取工作流的热度情况,在LSM树的每一层维护热度预测模型,对每个新生成的SSTable,通过预测其热度来分配不同粒度的布隆过滤器,然后使用特定缓存管理方案来维护缓存中的过滤器数据并处理工作流热度发生改变的情况.

2　相关工作

现代基于LSM树的存储架构多使用SSTable作为磁盘的存储组件,其最初提出于BigTable[5]. 图1展示了以LevelDB为例现代LSM树的设计模式,内存中的组件包括MemTable, Immutable MemTable;外存中的重要组件为SSTable.

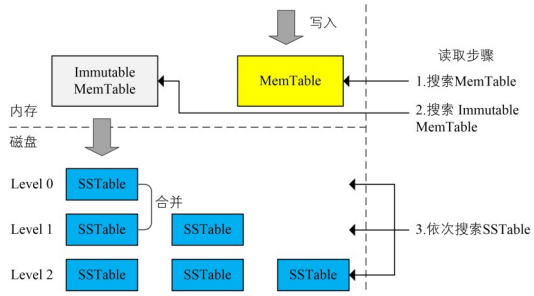

图1　以LevelDB为例的现代LSM树的整体架构

LSM树在写入时先将数据保存在内存的MemTable. 当MemTable过大时,LevelDB会对其产生一个静态快照Immutable MemTable,同时生成一个新的MemTable来接收最近的更新. 后台调度线程负责将这个快照保存在磁盘的Level 0中作为SSTable进行持久化存储. 当磁盘上某一层的文件大小达到一定规模时,系统会将其与下一层的SSTable进行合并,新生成的SSTable驻留在下一层中. 下一层的容量是上一层的固定倍数.

读取时内存组件存储数据相较于SSTable更新,因此会先被检索,然后按照层次顺序由低到高依次检查磁盘上的SSTable. 由于访问磁盘的时间代价较大,因此会采用一些优化手段来进行优化,如读取SSTable的页面前会使用过滤器来减少无效I/O.

访问的偏斜性在KV数据库中十分明显,大部分数据访问集中在少量SSTable. 因此在Monkey[6,7]中,作者为更低层的过滤器分配更细的粒度,直观上理解,访问会从低层到高层直到找到数据后停止,因此为低层分配更细粒度的过滤器起到的作用更大. ElasticBF[8]中分析了在数据的读取过程中,不同层间的SSTable的访问热度差距很大,即便是在同一层,不同SSTable的访问热度差距也很大,因此提出一种弹性过滤器方案:利用布隆过滤器的可分割性将过滤器分为若干个过滤器单元,并通过SSTable的访问热度动态调整内存中的过滤器单元. 这种方法牺牲了空间性能,尤其是对于一些小型键值对,过滤器元数据的空间开销将庞大到无法接受的地步[9]. 另外,热度继承导致SSTable初始热度值与实际热度值存在偏差,从而导致过滤器的小型过滤器单元在内存与外存之间频繁换入换出,这对系统性能也会造成影响.

3　HBF-LSM框架

图2展示了本文提出的HBF-LSM的结构. 在数据的读取阶段,Hotness Manager将会记录每一层的每个SSTable在一个最近时间窗口内的访问热度,并由Model Manager来训练每一层的热度预测模型. 当由于内存组件的刷新或者外存上SSTable合并而产生新的SSTable时,Model Manager将会根据该SSTable的数据分布,利用每一层模型来预测访问热度,新SSTable的过滤器粒度将由预测的热度分级决定. 同时,Model Manager将对其所维护的每一层的模型以一定时间间隔进行重新训练以保持精确度,防止热度预测产生较大误差影响过滤器的粒度分配. Cache Manager负责维护固定大小的内存缓存区中的过滤器数据,通过多级优先队列保证缓存区中始终为热度较高的过滤器.

3.1　Hotness Manager:热度定义与分级

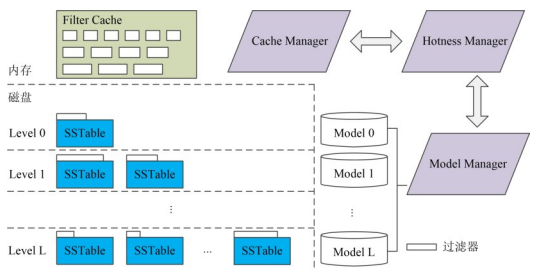

图2　使用异构布隆过滤器的LSM树结构(HBF-LSM)

本文将一系列Get请求当成一条逻辑上的时间轴,而最近时间窗口代表了与当前Get请求最近的若干个请求,其长度为u. 假设LSM树有L+1层(0层~L层),且第l层的SSTable数量为$N(l)$,SSTable总数量为N.

首先需要根据SSTable的访问情况定义热度. 第l层的第k个SSTable在时间点t上的访问情况$\mathrm{access}_{l,k}(t)$

定义为

$$\text{access}_{l,k}(t)=\begin{cases}1, \text{SSTable is accessed}\\0, \quad \text{else}\end{cases} \tag{1}$$

假设当前时间点为c,则第l层的第k个SSTable的当前热度$f_{l,k}$定义为

$$f_{l,k}=\sum_{t=c-u+1}^{c}\text{access}_{l,k}(t)\cdot K(t-c) \tag{2}$$

其中$K(x)$可以理解为随时间衰减的权重函数. 在具体实现中,文本采用高斯函数的前半段:

$$K(x)=\frac{1}{\sqrt{2\pi\cdot\sigma}}\exp\left(-\frac{x^2}{2\cdot\sigma^2}\right),x\leqslant 0 \tag{3}$$

式中σ可自行调整. 给定$K(x)$为式(3),当某个SSTable在最近时间窗口内的每个时间点t都被访问过时,该SSTable的热度取得最大值$f_{\max}$.

接着需要根据某个SSTable的热度值决定需要给该SSTable分配的过滤器粒度. 将热度区间F划分为n个连续区间,即$F=\{\pi_1,\pi_2,\cdots,\pi_n\}$,$n$为热度分级数目. 与其对应的是某个热度区间所分配的过滤器粒度$G=\{g_1,g_2,\cdots,g_n\}$,即具有热度$f_{l,k}$的SSTable所分配的过滤器粒度应该为

$$\text{gran}\left(f_{l,k}\right)=g_i,f_{l,k}\in\pi_i \tag{4}$$

假设系统分配给过滤器的缓存区大小固定为B_f,平均每个SSTable能容纳的数据项为V,并且内存中能够驻留所有布隆过滤器,则在静态分配的情况下,在缓存区的每个过滤器分配到的位数可以表示为

$$g_s=\frac{B_f}{V\cdot\sum_{l=0}^{L}N(l)} \tag{5}$$

若一个工作流对每一层的SSTable $S_{l,k}$访问了$m_{l,k}$次,那么在静态分配下,由于假阳性率而造成的磁盘I/O次数为

$$P_s=\sum_{l=0}^{L}\sum_{k=1}^{N(l)}m_{l,k}\cdot(2^{-\ln 2})^{g_s} \tag{6}$$

而在本文提出的异构策略中,假设粒度为g_i的过滤器数量为$N_f(i)(i=1,2,\cdots,n)$,则由于假阳性率而造成的磁盘I/O次数为

$$P_h=\sum_{l=0}^{L}\sum_{k=1}^{N(l)}m_{l,k}\cdot(2^{-\ln 2})^{\text{gran}\left(f_{l,k}\right)}=\sum_{i=1}^{n}c_i\cdot(2^{-\ln 2})^{g_i}$$

$$\text{subject to:}\quad \sum_{i=1}^{n}N_f(i)=N,V\cdot\sum_{i=1}^{n}N_f(i)\cdot g_i\leqslant B_f \tag{7}$$

c_i表示粒度为g_i的SSTable被访问的次数. 第一个约束条件为数量约束,第二个约束条件为空间约束. 为了得到P_h的最优解,需要让c_i与$N_f(i)$建立联系. 这里假设c_i与$N_f(i)$成反比,即访问次数越多的SSTable在系统中存在的数量应该越少. 在热度明显的读取工作流中,只有小部分的SSTable会被工作流频繁访问到,因此这种假设也是符合直观的. 根据柯西不等式和第一个约束,最优的$N_f(i)$取值可表示为

$$N_f(i)=\frac{(2^{-\ln 2})^{g_i}}{\sum_{j=1}^{n}(2^{-\ln 2})^{g_j}}\cdot N \tag{8}$$

式中等号右侧左半部分可以看作各粒度过滤器在SSTable总数量中的占比. 当使用n=6,G={20,16,12,8,6,3}的配置时,$P_s/P_h\approx 30$,即使用异构分配策略下由于布隆过滤器的假阳性而造成的无效磁盘I/O要比静态分配策略少得多.

最近时间窗口只关注局部时间内SSTable的热度情况,事实上当窗口长度u无限大时,$f_{l,k}$与$m_{l,k}$正相关,即访问次数越多的SSTable热度越高,个数越少,那么理论上F可以均匀划分为

$$\left\{\pi_i\in\left[f_{\max}\cdot\frac{i-1}{n},f_{\max}\cdot\frac{i}{n}\right)\right\},i=1,2,\cdots,n \tag{9}$$

在实现中,可以先用式(9)初步确定其粒度,然后用式(8)判断该粒度下是否还能容纳其他过滤器,若不能,则将过滤器的粒度降低一级,以此类推. 这样既能使整体假阳性率尽可能低,又能保证过滤器的容量限制在静态分配所需容量之内.

3.2 Model Manager:模型的训练与维护

如前文所述,LSM树中的SSTable访问在层与层之间表现得十分不平衡,因此本文按照层的粒度来构建多个模型,防止只采用一个模型而造成较大误差. 在数据分布的表示中,将整个键的空间划分为e个大小相同且连续的区间$\{h_1,h_2,\cdots,h_e\}$,那么某个SSTable的数据分布可以表示为行向量$\boldsymbol{r}=[d_1,d_2,\cdots,d_e]$,其中$d_i$表示键在$h_i$范围内的数据数量. 那么问题就可以转化为使用数据分布$\boldsymbol{r}$来预测热度值f. 考虑到LSM树中的高实时性要求,本文使用训练速度较快的线性回归来构建每一层的模型$M_l(l=0,1,\cdots,L)$.

我们的目标是在LSM树的每一层中构建模型M_l,使得模型能够通过该层中新生成SSTable的数据分布来预测该SSTable在未来一段时间内的访问热度$f_{l,k}$. 即目标是求$M_l(\boldsymbol{r}_{l,k})=\boldsymbol{r}_{l,k}\cdot\boldsymbol{w}_l+b_l$,使$M_l(\boldsymbol{r}_{l,k})\approx f_{l,k}(k=1,2,\cdots,N(l))$,其中$\boldsymbol{w}_l$为长度为$e$的列向量,$b_l$为标量. 每一层的代价函数$J_l$表示为

$$J_l\left(\boldsymbol{w}_l,b_l\right)=\sum_{l=0}^{L}\sum_{k=1}^{N(l)}\left(M_l\left(\boldsymbol{r}_{l,k}\right)-f_{l,k}\right)^2 \tag{10}$$

线性回归模型的优势在于可以在常数时间内得到最优的参数解,这个特点对于写入性能和读取性能要求都较高的LSM树至关重要.

3.3 Cache Manager:缓存维护与热度更新

本文使用多级优先队列来对内存缓存区中的过滤

器数据进行管理. 缓存队列分为 n 个级别,表示为 Q_1~Q_n,对应了 n 个过滤器粒度的分级,队列 Q_i(i=1,2,…,n)负责维护粒度为 g_i 的过滤器数据. 当一个过滤器粒度为 g_i 的SSTable被访问时,将该过滤器数据添加队列 Q_i,队列内以SSTable的热度作为优先级. 如果该过滤器早已存在于缓存中即缓存命中,则按照其新的热度值对其在队列中的位置进行调整. 而当缓存区的容量已满又需要调入新的过滤器数据时,需要将缓存区内的过滤器数据进行调出以容纳新的过滤器数据. 很显然由于队列间的优先度严格按照过滤器的粒度进行划分,粗粒度的过滤器数据将优先从缓存区中调出以容纳新的过滤器数据.

另外SSTable的热度值变化过大而不进行任何调整会严重影响系统性能,如热度上升时,原本过滤器粒度较粗,未来对该SSTable的访问调用过滤器进行键的存在性判断时,由于假阳性率过大而导致额外的磁盘I/O. 而某个SSTable热度下降时虽然不会对额外磁盘I/O有较大影响,但是会浪费有限的过滤器缓存空间. 因此对每个过滤器数据引入一个辅助字段I_LEVEL表示其初始所在的队列号,其值在SSTable生成的时候初始化并且不会发生变化. 当一个发生磁盘读取的Get请求结束时,对所有发生队列间移动的过滤器数据做如下操作:假设某SSTable当前所在队列等级为C_LEVEL,若某个SSTable满足:

$$|\text{C\_LEVEL} - \text{I\_LEVEL}| \geqslant \lfloor n - 1 \rfloor / 2 \qquad (11)$$

时则认为需要对该SSTable的热度进行修改,故将该SSTable数据其加入修改队列 Q_m,等待系统对其重构. 原本的过滤器数据仍然可以用于重构完成前的Get请求直到重构完毕. 重构时需要从磁盘读取SSTable的所有数据,修改过滤器粒度并写回磁盘并插入缓存. 图3展示了Cache Manager的缓存管理方案. 图中组件 S_0 的过滤器数据初始时位于队列 Q_1,在某个Get请求后位于队列 Q_2,此时不必立即修改过滤器粒度. 当另一个Get请求导致 S_0 的过滤器数据位于 Q_6 时,将 S_0 组件信息添加到 Q_m 等待重构.

4　实验评估

本文在LevelDB上实现HBF-LSM,并对其在热度稳定和热度变化的工作流进行了性能评估. 实验平台为8核Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz,内存为16GB,操作系统为Ubuntu 18.04 LTS. 默认设置block缓存为0并开启DirectIO,使用YCSB[10]作为性能测量工具.

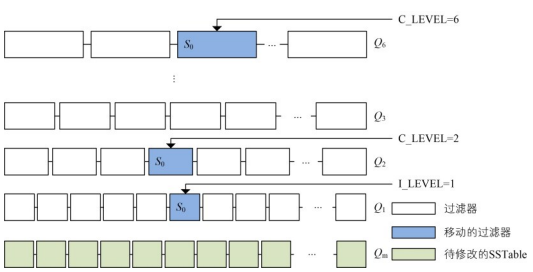

图3　HBF-LSM中的缓存管理方案

YCSB性能测试工具默认在生成数据库时,以递增顺序生成指定数量的键和随机生成的值插入数据库中,这将导致LSM树结构中每一层的各个SSTable范围相互独立,其他层的访问直接被SSTable的元数据信息过滤掉. 因此提醒读者在测试过滤器性能时,可以将整个键的空间进行随机打乱再插入数据库中,或者是在Key Space的范围内随机挑选键插入数据库中.

在数据库的建立过程中,将10%的读取工作流均匀加入到写入工作流中作为混合工作流以学习数据的热度,而剩下90%的读取工作流将作为只读工作流. 键值对的大小设置为1KB,数据库最终大小为83GB(100M个键值对,约1330个SSTable).

对于Get请求,默认产生10M个符合Zipfian分布的请求,这些请求在部分键上的热度高于其他键. 为发挥布隆过滤器的作用,默认设置了 p=0.7(p 为zero lookup的查询占比)的请求来访问不存在的键. Zero lookup在实际查询中很常见[11],如在执行insert if not exist等操作时.

首先对热度稳定的工作流进行测试以验证Hotness Manager与Model Manager的作用. HBF-LSM的主要计算开销体现在数据写入时的模型更新与热度计算上,数据读取阶段的计算开销与LevelDB等分层式LSM架构一致. 因此为了评估HBF-LSM的计算开销,本文也统计了写性能的变化情况. 表1是数据库构造时接触10%的读取工作流时在HDD和SSD下的实验结果,可以看到HBF-LSM能够学习到读取工作流的热度情况并合理分配布隆过滤器的粒度,提升了整体的读取吞吐量(HDD下约45%,SSD下约49%),大幅降低了读取操作的延迟,同时写入吞吐量几乎没有受到影响.

表2是在数据库上执行剩下的90%的只读工作流得到的结果,与混合工作流下相差不大,相较于前者在读取吞吐量上稍有提升(未开启block缓存时约53%,开启block缓存时约22%). 从平均查询I/O的数目可以发

表1　混合工作流下性能测试实验结果

配置		吞吐量(ops/s)	延迟(ms)	写入吞吐量(ops/s)
LevelDB	HDD	167	5.96	**767**
HBF-LSM		**245**	**4.08**	753
LevelDB	SSD	3279	0.31	**802**
HBF-LSM		**4884**	**0.20**	783

现相比于LevelDB固定分配的方式,HBF-LSM减少了绝大部分无效磁盘I/O(未开启block缓存时约64%).开启block缓存后读取吞吐量提升效果下降较多(约22%),平均查询I/O数在HBF-LSM下甚至已低于"必要磁盘I/O",这说明只读工作流中,缓存机制对LSM树结构的性能影响较大.

为了验证Cache Manager的作用,我们接着使用了热度变化的工作流来构造数据库,接触的读取工作流

表2 只读工作流下性能测试实验结果

配置		吞吐量(ops/s)	延迟(ms)	平均查询I/O
LevelDB	关闭block缓存	167	5.96	1.09
HBF-LSM		**257**	**3.88**	**0.39**
LevelDB	开启block缓存	270	3.7	0.65
HBF-LSM		**330**	**3.0**	**0.28**

前30%与后70%分别来自采样于两个不同的Zipfian分布,得到了表3的结果.我们统计了因为热度变化而发生重构的SSTable数量为137个,约占构造完总数量的10%.可以发现相比于热度稳定的工作流,由于触发了过滤器粒度调整以及使用粒度不匹配的过滤器数据使得吞吐量略有下降,但整体上依然高于固定分配方式(约39%).

最后将HBF-LSM与Monkey和ElasticBF进行对比.根据文献[6]中4.1节可以计算出每一层的最优假

表3 热度变化工作流下性能测试实验结果

配置	吞吐量(ops/s)	延迟(ms)	平均查询I/O
LevelDB	168	5.96	1.09
HBF-LSM	**235**	**4.26**	**0.46**

阳性率和布隆过滤器位数,然后在LevelDB上实现Monkey的分配策略.对于ElasticBF采用与本文相同的配置.实验中过滤器占用内存大小限制为256MB.实验数据如下:1G个16B的数据项,64M次符合Zipfian分布的读取请求.HBF-LSM和Monkey占用13GB左右的外存,而ElasticBF则占用14.5GB左右,牺牲了一部分外存性能.最后在只读工作流下调整zero lookup查询占比p以比较它们的查询性能,表4为得到的结果,可以发现,HBF-LSM在不同p值下性能都要超过Monkey和ElasticBF.如前文所述,ElasticBF在生成新SSTable时,使用父SSTable的热度平均值作为初始热度,可能与实际热度值存在偏差.这将导致有限的过滤器缓存空间浪费,并且需要额外的磁盘I/O进行过滤器单元内外存交换来弥补初始偏差.而Monkey很多时候无法实现理论上的性能,原因有两点:(1)Monkey针对均匀工作流进行优化,而现实中的工作流多含有偏斜性;(2)实际LSM结构的数据库各层的容量不会完美符合理论上的容量放大比率,而且会伴随着数据库的构建而发生很大的变化,Monkey的"最优分配方式"通常并不是最优.

表4 与Monkey和ElasticBF对比实验结果

配置	不同p下的平均查询I/O					
	p=0.5	p=0.6	p=0.7	p=0.8	p=0.9	p=1.0
HBF-LSM	**0.56**	**0.46**	**0.38**	**0.27**	**0.18**	**0.08**
Monkey	0.74	0.62	0.50	0.40	0.28	0.17
ElasticBF	0.62	0.50	0.43	0.32	0.22	0.11

5 结论

本文针对LSM树中布隆过滤器假阳性率与空间占用(包括内存和外存)的矛盾,提出了基于访问热度分配过滤粒度的HBF-LSM结构,并在LevelDB上进行了实现.实验证明本文所提出的结构在相同的外存占用与内存消耗下,提升了LSM树在热度明显的工作流下的读取性能.

参考文献

[1] O'Neil P, Cheng E, Gawlick D, et al. The log-structured merge-tree (LSM-tree) [J]. Acta Informatica, 1996, 33(4): 351 – 385.

[2] Dayan N, Idreos S. The log-structured merge-bush & the wacky continuum[A]. Proceedings of the 2019 International Conference on Management of Data[C]. New York, USA: ACM, 2019. 449 – 466.

[3] Luo C, Carey M J. LSM-based storage techniques: a survey [J]. The VLDB Journal, 2020, 29(1): 393 – 418.

[4] Bloom B H. Space/time trade-offs in Hash coding with allowable errors[J]. Communications of the ACM, 1970, 13 (7): 422 – 426.

[5] Chang F, Dean J, Ghemawat S, et al. Bigtable: a distributed storage system for structured data[J]. ACM Transactions on Computer Systems (TOCS), 2008, 26(2): 1 – 26.

[6] Dayan N, Athanassoulis M, Idreos S. Monkey: optimal navigable key-value store[A]. Proceedings of the 2017 ACM International Conference on Management of Data[C]. New York, USA: ACM, 2017. 79 – 94.

[7] Dayan N, Athanassoulis M, Idreos S. Optimal bloom filters and adaptive merging for LSM-trees[J]. ACM Transactions on Database Systems (TODS), 2018, 43(4): 1 – 48.

[8] Li Y, Tian C, Guo F, et al. ElasticBF: elastic bloom filter with hotness awareness for boosting read performance in large key-value stores[A]. Proceedings of the 2019 USE-

NIX Annual Technical Conference[C]. Berkeley, USA: USENIX Association, 2019. 739 – 752.

[9] Wu X, Xu Y, Shao Z, et al. LSM-trie: an LSM-tree-based ultra-large key-value store for small data items[A]. Proceedings of the 2015 USENIX Annual Technical Conference[C]. Berkeley, USA: USENIX Association, 2015. 71 – 82.

[10] Cooper B F, Silberstein A, Tam E, et al. Benchmarking cloud serving systems with YCSB[A]. Proceedings of the 1st ACM Symposium on Cloud Computing[C]. New York, USA: ACM, 2010. 143 – 154.

[11] Bronson N, Amsden Z, Cabrera G, et al. TAO: Facebook's distributed data store for the social graph[A]. Proceedings of the 2013 USENIX Annual Technical Conference[C]. Berkeley, USA: USENIX Association, 2013. 49 – 60.

作者简介

俞加平　男,1995年生于浙江湖州.现为宁波大学信息科学与工程学院硕士研究生.主要研究方向为非关系型数据库、数据挖掘等.
E-mail:1811082061@nbu.edu.cn

钱江波　男,1974年生于浙江宁波.现为宁波大学教授、博士生导师.主要研究方向为数据库管理、多维度索引及查询优化等.
E-mail:qianjb@163.com

陈华辉(通信作者)　男,1964年生于浙江宁波.现为宁波大学教授、博士生导师.主要研究方向为数据流处理及大数据处理、数据挖掘等.
E-mail:chenhuahui@nbu.edu.cn

董一鸿　男,1969年生于浙江宁波.现为宁波大学教授、硕士生导师.主要研究方向为大数据处理、数据挖掘及人工智能.
E-mail:dongyihong@nbu.edu.cn